Spacecraft Command Verification: The AI Solution

Lorraine M. Fesq
Amy Stephan
TRW
Redondo Beach, CA

Brian K. Smith
UCLA
Los Angeles, CA

ABSTRACT

Recently, a knowledge-based approach was used to develop a system called the Command Constraint Checker (CCC) for TRW. CCC was created to automate the process of verifying spacecraft command sequences. To check command files by hand for timing and sequencing errors is a time-consuming and error-prone task. Conventional software solutions were rejected when it was estimated that it would require 36 man-months to build an automated tool to check constraints by conventional methods. Using rule-based representation to model the various timing and sequencing constraints of the spacecraft, CCC was developed and tested in only three months. By applying artificial intelligence techniques, CCC designers were able to demonstrate the viability of AI as a tool to transform difficult problems into easily managed tasks. This paper discusses the design considerations used in developing CCC and examines the potential impact of this system on future satellite programs.

INTRODUCTION

Even after a spacecraft is launched, it continues to receive information from ground stations telling it what actions to perform. This information is in the form of spacecraft commands. These commands are formulated on the ground, transmitted to the spacecraft, and used to instruct the spacecraft to perform actions such as turning instruments on and off, switching relays, or maneuvering the craft into a new orientation. The spacecraft hardware being commanded consists of highly specialized electronics which must be carefully reconfigured. The commands sent to the spacecraft must follow strict guidelines as to their order and timing. These guidelines are known as constraints, and all sequences of commands must be examined to assure that they meet all constraints before they are transmitted to the spacecraft. This pre-checking of command sequences is a very involved and time-consuming task. Performed manually, it could take a week to check one set of commands. Developing a conventional software program to automate this process also would be a difficult and costly task, estimated to take at least 36 man-months. This paper describes an expert system that was developed in only three months to automate the process of checking spacecraft command sequences for constraint violations.

DESCRIPTION OF THE PROBLEM

There are two major types of constraints which must be met within all spacecraft command sequences: *timing* constraints and *ordering* constraints. Consider the following example. A spacecraft is commanded to start recording information onto an on-board tape recorder. This action may involve numerous individual commands, many of them going to the same box on the spacecraft. Each box has a limit of how fast it can receive commands, in much the same manner as humans have a limit of how fast they can absorb sensory input. If commands are sent too fast, the box will not receive all of them, and the spacecraft will not perform the desired action. For example, all commands to the spacecraft Tape Recorder Box must be separated by 50 milliseconds. This is an example of a *timing* constraint.

To accomplish the task of recording data, a number of commands must be sent to the Tape Recorder, and these commands must be in a specific order. An example of an *ordering* constraint would be that the "Tape Recorder On" command must precede the "Tape Recorder Rewind" command, which in turn must precede the "Tape Recorder Record" command.

Spacecraft command sequences can contain hundreds of commands, all of which must meet all timing and ordering constraints. The number of constraints can also be on the order of hundreds. Checking all commands (n) against all constraints (k) would require (n(n-1)/2)*k operations, or order $O(n^2k)$ operations. Checking a medium sized file of 50 commands against 50 constraints could require up to 61,250 operations - an enormous task when performed by hand. An automated system seems an appropriate solution, but developing a conventional software package to perform the checking has been estimated on one spacecraft program to be prohibitive.

The following section describes an AI solution to automating this process, and contrasts the solution to a less efficient, more costly conventional approach.

AI SOLUTION TO THE PROBLEM

The constraints against which commands are checked closely resemble expert system rules. Constraints generally consist of several conditional clauses and an error that will occur if these conditions are met. This format can be easily translated into the type of rules typically used in rule-based production systems. For example:

```
IF    the [tape-recorder rewind] command is received before the
          [tape-recorder on] command
THEN  [constraint X is violated]
```

Commands themselves are symbolic in nature, and generally can be represented as a spacecraft box and an action to be performed on that box. It is more natural to think of a command using its symbolic representation, i.e. [TAPE-RECORDER ON], than its numeric (hexidecimal) representation. Because commands are easily represented as sets of symbols, they can be used as facts in an expert system.

A commercial inference engine could efficiently match facts against rules and note violations, leaving the designer the tasks of writing the rules, converting commands into facts and choosing the appropriate hardware and software tools with which to build the system. An off-the-shelf inference engine seemed the ideal choice for this straightforward production system. By employing such a system, we would have access to efficient unification and database management algorithms, leaving the designer free to concentrate on optimizing the rule and fact representation.

Expert systems seemed ideally suited to solve what had been an unsolvable problem. The set of constraints on satellite commands, although large, is well-documented and would require a domain expert only to explain the highly specialized language in which these constraints are described. Programming an expert system to check constraints would require a straightforward conversion of constraints into rules and a scheme for representing commands as facts. To code this system conventionally would require pages of awkward IF-THEN and CASE statements, performing numeric calculations on data that is inherently symbolic. In addition to the reduction in code size, the expert system approach also promised a significant increase in speed. The inference engine would at most match each command against each constraint clause, an operation of order O(nk) as opposed to order $O(n^2k)$, significantly reducing the time needed check a file of commands.

Within three months, an expert system called the Command Constraint Checker (CCC) was developed (see Figure 1). This system runs on an IBM-compatible PC, which is currently used in NASA Control Centers to write and store command procedures

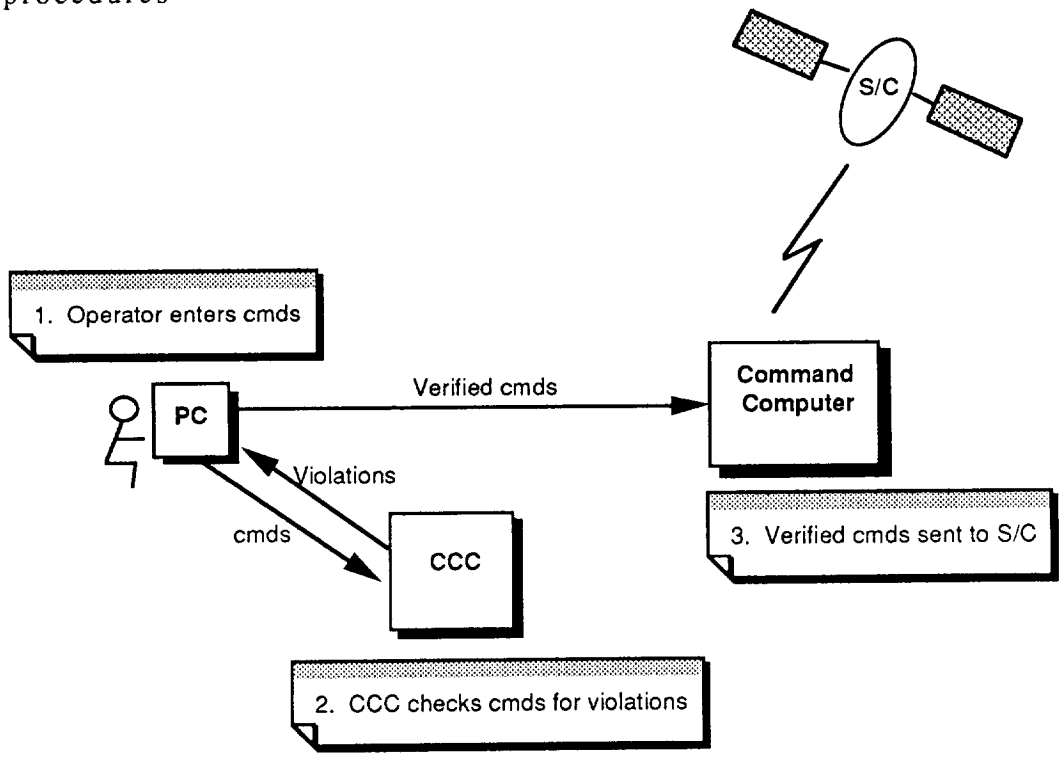

Figure 1. Automated Process of Verifying Spacecraft (S/C) commands

before sending them to a spacecraft. Housing the CCC on the PC allows the program to process actual command files to be sent to a spacecraft, eliminating the need for users to input lengthy command sequences into the CCC. Running the expert system on the PC also allows the CCC to be used on-line; command sequences can be checked immediately before being transmitted.

The Command Constraint Checker consists of a menu-driven user interface, a set of procedures to parse various formats of command files into rules, a rule base, a commercial inference engine and a mechanism for reporting violations to the user. To limit the number of rules in the system, rules are designed to be as general as possible. For example, a constraint might read, "Commands to the tape recorder must be at least 50 ms apart." In the format used in command procedures, however, it is not always easy to recognize which commands control the tape recorder. A set of commands to this instrument might be encoded as follows: CMT1ON, CMT2ON, CMT1OFF, CMT2OFF, CMRWND, CMFF. Using these mnemonics, fifteen rules would be needed to assure that none of the potential tape-recorder command pairs were less than 50 ms apart. If the command designation, TAPE-RECORDER, were included in each command fact, only one rule would be needed to ensure that all tape recorder commands were properly spaced. Using an existing database containing information about all valid spacecraft commands, we were able to abstract command mnemonics into facts amenable to the more general language of the command constraints. As mnemonics are read from a command file, they are matched against the command database and the following information is asserted as part of the command fact:

COMMAND MNEMONIC	As it appeared in the command procedure
HEX CODE	The actual hex representation of the command sent to the spacecraft.
DESTINATION BOX	The box that will receive this command, i.e. tape recorder.
COMMAND DATA	The action to be sent to this box, i.e. REWIND

If a command mnemonic is not found in the database, the command is illegal and this information is included in the user's error report. A routine to optimize the database command mnemonic search is included in the CCC.

The CCC contains parsing routines to convert several types of command procedures into command facts. A menu-driven user interface allows the user to input a command file name, a command database name, the rate at which commands will be sent to the spacecraft and the type of command file to be parsed. The command file formats range from simple lists of commands and WAIT statements, to complicated files containing IF-THEN-ELSE, GOTO and WAIT statements as well as variables. CCC employs the appropriate routines to parse the command file into a list of mnemonics, checks these mnemonics against the database and assigns an absolute time to each command. The CCC bases this absolute time on the user-supplied rate at which commands are being sent to the spacecraft. The first command is given the absolute time of 0, and each successive command is assigned a time based on the uplink rate and the number and duration of WAIT statements in the command file. The line number of the command in the input file also is part of the command fact. If the command fact causes a violation, this line number is included in the user's

error report, allowing her to easily edit the original command file. A command fact with all its fields might look like this:

```
[CMT1OFF     TAPE-RECORDER     OFF   133A077     .128              5]
mnemonic     destination       data  hex code    absolute time     line
```

While CCC automatically converts commands into facts, the designer must represent constraints as rules. The task of maintaining the rule base is simplified by several factors. All constraints on commands for a given satellite typically are well-documented in that satellite's mission operations handbook. The page in the handbook on which a constraint appears is included in that constraint's documentation. This allows the engineer in charge of maintaining the system to easily delete a rule if the constraint it represents is later deemed unnecessary. Conversely, the close correlation between IF-THEN rules and the language in the constraint descriptions will allow an engineer with little or no background in expert systems to add new constraints to the rule base.

The CCC required a compact forward-chaining inference engine capable of interfacing with a conventional language and running on a PC. CLIPS, a NASA-built expert system shell written in C meets all of these needs. Since CLIPS rules can be run from within a C program, we were able to embed the expert system in a conventional C program. This allowed us to perform procedural tasks, such as the user-interface and file operations, in C and actually identify the constraint violations using CLIPS (Figure 2). After all inferences have been made, control returns to the C program which records the CLIPS violation facts in a readable file for the user.

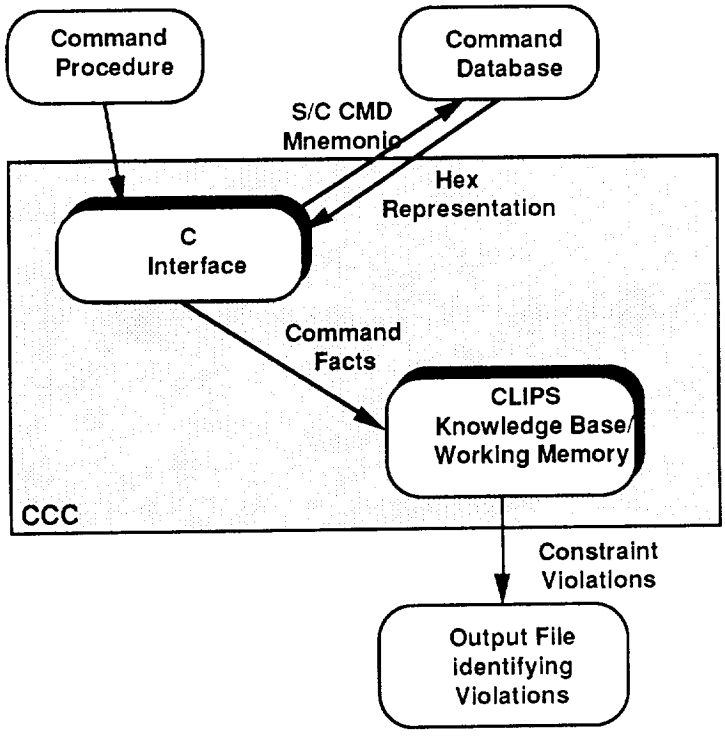

Figure 2. CCC Block Diagram

STATUS

After three months of coding, a working version of the CCC is ready to be delivered to its users. The program solicits user input through a menu-driven interface, parses three formats of command files, checks these commands against a command database and asserts them as command facts. It then runs its rule base of 70 rules against the facts, and provides the user with a readable output file listing the nature of all violations and warnings, including the line number in the original command file of each flagged command. This information allows the user to edit her command procedure and re-check it with CCC before sending the commands to the spacecraft.

A rule base for a specific spacecraft, NASA's Gamma Ray Observatory (GRO), was developed to test the CCC. The 70 rules now in the CCC database cover about 90 percent of all GRO command constraints, including the constraints on those commands most often sent. A detailed user's guide has been developed, explaining how the CCC may be adapted to a new spacecraft, as well as how it may be modified to accommodate new constraints that arise during the life of the spacecraft. Potential users have been given the opportunity to experiment with CLIPS and gain familiarity with its rule structure so that they can maintain the rule base with little aid from the designers. The system is able to check a typical file of 50 commands in five minutes. This is a considerable savings over the hours that previously were spent checking command sequences manually.

No major problems were encountered while developing CCC. The choice of languages and the hardware platform proved easy to use and adequate for our needs. Excellent cooperation from domain experts and potential users sped the development of this system, allowing us to complete the project on schedule and within budget.

EFFECTIVENESS OF IMPLEMENTATION

Response from users has been enthusiastic. In several beta tests and demonstrations, users have been impressed by the performance and usability of CCC. Engineers who plan to apply this tool are happy to be free of the tedious chore of checking command sequences by hand. Management also is pleased with the results of the project, which produced an automated tool for less than 10 percent of the estimated cost of building such a system using conventional methods.

CCC is a classic example of how examining a software problem from an AI perspective can change the nature of the problem. By observing the symbolic nature of the commands and constraints, the CCC designers were able to transform a difficult conventional problem into a relatively straightforward expert system task. The application of artificial intelligence techniques to this problem produced a useful tool that will save many wasted hours, thousands of dollars and potentially the life of a spacecraft.

REFERENCES

1. Fesq, L., & Stephan, A. (February 1989). Advances in Spacecraft Autonomy Using Artificial Intelligence Techniques. *12 Annual AAS Guidance and Control Conference*, Keystone, Colorado.

1. Giarratano, Joseph C. (1988). *CLIPS USER's Guide*. Artificial Intelligence Section, Lyndon B. Johnson Space Center.

2. Knuth, Donald E. (1973). *The Art of Computer Programming*, Vol. 1. Addison-Wesley Publishing Co., Reading, Massachusetts.

3. Nilsson, Nils J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufman Publishers, Inc., Los Altos, CA.

4. Waterman, Donald A. (1986). *A Guide to Expert Systems*. Addison-Wesley Publishing Co., Reading, Massachusetts.